Article:
Eberle A. Rambo, Christoph Seitz, Selma Saidi, Rolf Ernst, "Bridging the Gap between Resilient Networks-on-Chip and Real-Time Systems", in IEEE Transactions on Emerging Topics in Computing , vol.PP, no.99, pp.1-1.

https://doi.org/10.1109/TETC.2017.2736783

Bridging the Gap between Resilient Networks-on-Chip and Real-Time Systems

Eberle A. Rambo, *Student Member, IEEE,* Christoph Seitz, Selma Saidi, *Member, IEEE,* and Rolf Ernst, *Fellow, IEEE*

Abstract—Conventional fault-tolerance approaches for Networks-on-Chip (NoCs) cannot be applied to high dependability systems due to their different goals and constraints. These systems impose strict integrity, resilience and real-time requirements. In order to meet these requirements, all possible effects of random hardware errors must be taken into account, silent data corruption must be prevented and the resulting system must be predictable in the presence of errors. In this paper, we present a wormhole-switched NoC with virtual channels for high dependability systems hardened against soft errors. The NoC is developed based on results of a Failure Mode and Effects Analysis. It efficiently handles errors in different network layers and operates with formal guarantees. Our experimental evaluation, including an industrial avionics use case, shows that the network is able to achieve predictable behavior even in aggressive environments with very high error rates while presenting competitive overheads.

Index Terms—Network-on-Chip, Real-time, High Dependability, Reliability, Soft Errors.

1 Introduction

Multiprocessor Systems-on-Chip (MPSoCs) are currently being explored for high dependability systems [1]. When compared to single-cores, MPSoCs are more efficient and provide the performance required to implement increasingly complex functionalities demanded by the market. Examples in different industries are the Flight Management System (FMS), Advanced Driver Assistance System and Data Centers. Since a failure has serious safety and commercial consequences, such systems must meet strict requirements specified by contract and safety standards [2], [3], [4]. Despite their different target applications, these systems share an essential common feature: error detection.

Error detection is the base to any fault containment or tolerance technique. Only after the error is detected (or foreseen) can the system trigger a corrective (or proactive) measure. Therefore, error detection must be able to detect all relevant errors, must do so as soon as possible and with acceptable overhead. Otherwise, in case of silent data corruption, error containment and recovery strategies are rendered useless. When the corruption is finally detected, the error has already propagated and, even if the total error impact can be assessed, the recovery is beyond the capabilities of system, thereby resulting in a failure.

This work focuses on a main component of MPSoCs, the Network-on-Chip (NoC) [5]. NoCs are the scalable interconnect solution in MPSoCs, where memory and I/O operations are packets transmitted over a small-scale network. As the central interconnect, the NoC must be appropriately designed to detect and handle noise and errors that may occur in its infrastructure in order to provide dependable service [6], [7], [8]. Let us now point out the first requirement of a NoC for high dependability [8]:

- **Integrity**: no undetected error (silent data corruption) should be present in the system. Therefore, if the NoC delivers a packet, the packet and its delivery must be correct, i.e. errors in the NoC are detected and contained to the NoC.

Integrity at the component level is the minimum requirement to implement fail-silent or fail-operational failure strategies at the system level. Yet, it does not address the component or system reliabilities. That is, the system fails controllably but it may fail too often. In a high dependable system, a second requirement is identified:

- **Resilience**: the NoC must have the capability to recover from errors and to return to an operational state, tolerating possible data loss.

Random hardware errors can be of transient nature, called soft errors, or permanent nature, called hard errors [6], [9]. In this paper, we focus on soft errors. Soft errors can be abstracted as *bit-flips* in memory or in registers. Their sources are cosmic radiation, alpha particle strikes and process variability [9]. In addition, we consider crosstalk noise as a soft error [10]. Soft errors have different impacts depending on where and when they occur. In [6], [11], the authors show that, besides the standard effects usually considered in the literature, such as packet loss and derouting, and despite being caused by transient faults, soft errors can have static effects that lead to continuous corruption and blocking scenarios during runtime. Moreover, errors can propagate and indirectly affect the background traffic as

- *E. A. Rambo and R. Ernst are with IDA, Braunschweig University of Technology, Hans-Sommer-Str. 66, D-38106 Braunschweig, Germany. E-mail: rambo@ida.ing.tu-bs.de and ernst@ida.ing.tu-bs.de*
- *C. Seitz was with the IDA, Braunschweig University of Technology, during the development of this work.*
- *S. Saidi is with ES, Hamburg University of Technology, am Schwarzenberg-Campus 3 (E), D-21073 Hamburg, Germany. E-mail: selma.saidi@tuhh.de*

Manuscript received September 1, 2016; revised April 19, 2017.
This work has been partially funded by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 – spp1500.itec.kit.edu).

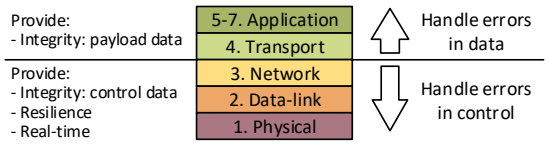


Fig. 1. OSI network model. Errors affecting the control of the network are handled in the lower layers. Errors affecting data/payload are addressed in the upper layers [14].

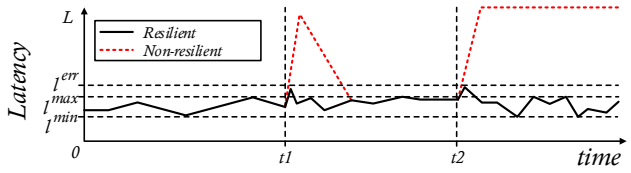


Fig. 2. Performance of a traffic stream in a resilient and a non-resilient NoC over time.

well. Such error effects cause the *violation* of integrity and resilience requirements and thus the failure of the system.

Besides integrity and resilience, certain applications require guarantees that the system will respond in time. In real-time systems, a subclass of high dependability systems, the system must respond in time even under errors, up to a maximum error rate. That means that the system must detect errors and recover within a determined interval of time. These are real-time/predictability requirements:

- **Real-time** requirements: the system must respond in time as well as recover in time to meet the target application's timing constraints. In the underlying hardware design, this requirement translates into *predictability*, where worst-case (minimum performance) guarantees must remain valid also in the presence of errors.

In this paper, we propose a NoC that satisfies strict integrity, resilience and real-time requirements. In contrast to the state of the art, we consider all possible impacts and durations of error effects [6]. We separate and handle the challenges in different network layers [12], as illustrated in Figure 1. Errors affecting the NoC's control logic and data are handled in the lower layers. This results in a highly available but lossy service to the upper layers, as error effects are restricted to packet corruption, loss or small delay. Guaranteed integrity and packet delivery of the transmitted data are selectively provided in the upper layers by means of Automatic Repeat reQuest (ARQ)-based protocols.

ARQ-based protocols operating in the transport layer have been recently formally analyzed for real-time networks [7], [13]. As shown by the authors, ARQ can be used without jeopardizing the predictability of the system. However, it does require the underlying network to limit the effects of errors and rule out static effects [7]. To achieve that, we exploit well-known fault containment and retransmission techniques together with the proposed resilient router design and resilient virtual channel (VC) flow-control. The VC flow-control manages the access to VCs in wormhole-switched NoCs. It is a major contributor to cases of static effect due to a dependency on the state of neighboring routers that only becomes evident in case of errors [6]. To overcome those issues, we propose a resilient VC flow-control. As a results, the proposed NoC operates under soft errors with formal guarantees [7].

The **contribution** of this paper is a wormhole-switched NoC for high dependability real-time systems. This is enabled by four key aspects in our design: **(1)** corrupt packet dropping policy enforced by a strong fault containment; **(2)** resilient router design; **(3)** resilient virtual-channel flow control for statically assigned VCs; and **(4)** reliable transmission. Our experimental results show that, even under very high error rates, the resilient NoC presents predictable behavior and is able to achieve it with acceptable hardware overhead. Although it focuses on the requirements of high dependability real-time systems [8], the approach can also be applied to general systems. To the best of our knowledge, this is the first NoC suitable for high dependability systems and able to provide formal guarantees under errors.

2 Related Work

Fault tolerance in Networks-on-Chip has been constantly researched throughout the years [15]. It has been explored at the link layer [16], [17] as well as the network layer [18], [19], [20], [21], [22]. Focusing on the requirements of general purpose and high performance computing systems, the approaches usually consider that soft errors cause packet corruption, loss or derouting, and that these effects are transient.

However, according to results of the Failure Mode and Effect Analysis (FMEA) introduced in [11] to meet certification requirements, later extended in [6], besides the standard effects usually considered in the literature, such as packet loss and derouting, soft errors can have static effects leading to continuous corruption and blocking scenarios during runtime. Moreover, errors can propagate and indirectly affect the background traffic [6], [11], [23]. Such error effects cause the *violation* of integrity, resilience and (real-time) predictability requirements and thereby the failure of the system.

In safety critical real-time systems, such errors have a fatal impact on the latency and, by extension, the predictability, leading to the failure of the entire system. Figure 2 illustrates the latency of a traffic stream observed over time on two different NoCs (retransmission is not included): a resilient predictable NoC; and a baseline NoC which is non-resilient and only predictable in the absence of errors. Both NoCs operate correctly (latency is within its l^{max} and l^{min} bounds) until soft errors occur at $t1$ and $t2$. In the resilient design, the effect is transient and its impact is bounded (l^{err}). In the baseline, soft errors cause static effects that result in very high latencies (L). After $t1$, which could be caused by the derouting of a packet from another traffic stream, the non-resilient design is able to recover. After $t2$, however, it remains blocked: affected packets accumulate in the routers' buffers and backpressure propagates throughout the network, leading to permanent blocking of the NoC. The NoC design must rule out static effects without triggering a network reset or handling them as hard errors and ensure controlled impacts (l^{err}).

The objective of existing approaches is to increase the overall reliability of the network. Most of the work [20], [21], [22] target packet-switched networks, providing reliability and guaranteed delivery of packets in the lower network layers based on hop-by-hop retransmission (between

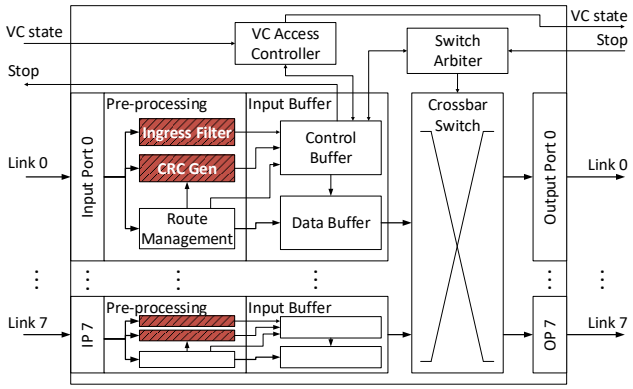


Fig. 3. The router architecture.

adjacent routers) [12]. Recovering from packet corruption and loss consists of retransmitting a correct copy from the previous router. The task of handling packet derouting is usually delegated to the dynamic routing, which delivers the packet through a different route after the packet has derouted. Variants with end-to-end retransmission and/or forward error correction and hybrids between both are also possible. In forward error correction [12], Error-Correcting Codes (ECCs) are employed to correct the data at the destination instead of retransmitting. Faster wormhole-switched networks have also been similarly addressed [18], [19].

The current state of the art in fault-tolerance for NoCs is well suited to increase the overall reliability of the network but do not satisfy the requirements of high assurance real-time systems. This is because the techniques either:

- Rely on dynamic routing [20], [21], [22]: the local routing decision violates the real-time requirements, as the predictability of the NoC is severely impaired. This is further aggravated by traffic deflection, which allows unexpected traffic overhead in the network. Predictability requires static routing [1], [24];
- Address packet-switched NoCs or wormhole-switched ones with dynamic VC allocation [18], [19]: traffic from different classes in real-time systems requires sufficient independence, i.e. isolation between each traffic classes, since they potentially have different treatment (e.g. priorities). This requires static VC allocation [1], [6];
- Use an insufficient error model [16], [17]: it has been shown that transient faults can lead to static effects [6], leading to unexpected blocking scenarios during runtime. All possible impacts and durations of error effects must be taken into account.

In order to tackle this problem, the NoC should be designed for high dependability systems [1], [8]. In the absence of errors, the NoC is predictable and satisfies real-time requirements. The NoC is then subjected to the FMEA-based analysis of [6]. The analysis uncovers all impacts of soft errors and classifies them with respect to duration and propagation, resulting in a comprehensive error model.

The rest of the paper is organized as follows. In the sequel, we present the baseline NoC and the error model, upon which this work is developed. Section 4 presents the hardening of the baseline NoC against soft errors. An evaluation of the resulting NoC is presented in Section 5. Finally, Section 6 draws the conclusions.

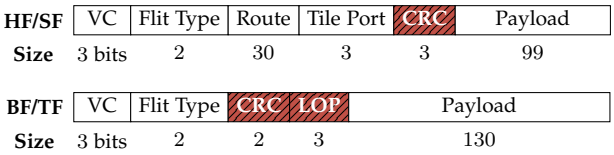


Fig. 4. Single (SF), Head (HF), Body (BF) and Tail (TF) flit formats.

3 Preliminaries

3.1 The Baseline Architecture

The starting point of this work is a typical NoC for real-time mixed-critical systems [1]. The network implements XY deterministic source routing, where the route and virtual channel (VC) are defined at the source; wormhole switching, where variable-sized packets are composed of fixed-sized Flow Control Units (flits); and virtual-channel flow control, where flits transit through a number of VCs. We assume the SLIP arbitration [25], a two-stage round-robin scheduler.

The router architecture is shown in Figure 3. The routers are input-buffered and the Input Buffer (IB) contains a FIFO queue for each VC. The Switch Arbiter (SA) implements the SLIP arbitration. The Virtual Channel Access Controller (VCAC) manages the access to VCs in downstream routers. The crossbar switch connects the input buffers to the respective output ports according to the arbitration grants. The flits are then forwarded from the input buffer directly to the downstream router. The router implements stop-and-wait flow control, raising a stop signal to inform the near-full state of the VC queues to the upstream router.

Route management is performed before the flit is stored in buffers to keep the routing data for the next router always in the first position. The route is encoded as a list of *runs*, which is a pair: output port (direction) and number of hops (distance). During a run, the hop counter is decremented. Once the packet finishes a run, the route is rotated: the finished run is moved to the end of the list and the next run becomes the head. Although we focus on the architecture, we assume the router execution spans a 4-stage pipeline.

The packets have variable size and are composed of a Head Flit (HF), zero or more Body Flits (BFs), and one Tail Flit (TF). A Single Flit (SF) packet is also supported. The flit formats are depicted in Figure 4. HFs and SFs (resp. BFs and TFs) are identical except for the flit type, which distinguishes their semantics. For transmission, a flit may be further divided in Physical Units (phits) to match the pipeline depth (i.e. 4 phits of 35 bits, not shown in Figure 4).

The shaded elements in the figures are not part of the baseline NoC. They are extensions required to implement the fault containment presented in this work. Ingress Filters and Cyclic Redundancy Check (CRC) Generators are added to the input ports of the routers cf. Figure 3. CRC codes and Last Output Port (LOP) data are added to the flit header cf. Figure 4. These additions will be detailed in Section 4.

3.2 The Error Model

We have derived a functional error model capturing all impacts of soft errors on the baseline NoC and their durations. The model is based on the analysis of [6], introduced in Section 2, and assumes single error scenarios. From the analysis of an implementation of the baseline NoC architecture, a

TABLE 1
Soft error effects per component

Component	**Fault Effects** on flits			
	Corruption	Loss	Derouting	Blocking
IB	✓	✓	✓	
Crossbar	✓	✓	✓	
SA		✓	✓	✓
VCAC				✓
Link	✓	✓		✓

fault occurring in the router or link that becomes a soft error may cause the following effects: flit corruption, flit loss, flit derouting, and flit blocking. Moreover, the effect depends on the affected component, as described in Table 1.

Flit corruption regards cases where either the flit header or the payload gets corrupted, but the flit reaches the router or network interface downstream. Flit loss regards the cases where the flit is lost before reaching the downstream router/network interface. Flit derouting accounts for cases where the flit is forwarded to the wrong router/network interface, whether the flit is corrupt or not. Blocking regards cases where the traffic is blocked at the router and cannot progress anymore. Notice that delayed transmission is included in this classification as a case of temporary blocking.

As mentioned previously, soft errors may cause static as well as transient effects. For instance, flit corruption may be a static effect when an error affects the pointer for a circular queue at the input buffer causing it to continuously read with the wrong offset; the blocking effect can be static when caused by a faulty VCAC, or transient when caused e.g. by losing the arbitration. This depends on the actual implementation and will be discussed in details in the next section. Static effects violate the *resilience* requirement since the system will not return to an operational state.

Last but not least, flit corruption and derouting may violate the predictability of the NoC, thereby violating the *real-time* requirements. This is essentially due to error propagation: a corrupt/derouted flit will affect other transmissions when arriving at the wrong router or VC queue [6]. This violates predictability in two ways: by causing the propagation of unexpected load through the NoC, and the indirect corruption of packets from other traffic streams.

Notice that the error model does not include errors in the configuration memory of FPGAs, assuming they are handled by techniques such as scrubbing [26].

4 HARDENING THE NOC

We harden the presented baseline NoC against the identified vulnerabilities to soft errors. These vulnerabilities are addressed by three mechanisms: Fault Containment (FC), Resilient Router Design (RR), and Reliable Transport (RT). These three mechanisms operate in different layers of the NoC and allow to address the high assurance systems requirements as shown in Table 2.

Let us first give an overview of the mechanisms. Fault containment is responsible for ensuring the integrity of the packets in the network. It is also responsible for containing the error to the affected router, preventing its propagation through the network and ensuring the predictability of the NoC. The policy for fault containment is packet dropping.

TABLE 2
Requirements addressed in the different layers of the NoC

OSI Layer	**Requirements**		
	Integrity	*Resilience*	*Real-time*
4-7. Transport - App.	RT		
3. Network	FC		FC
2. Data-link	FC	RR	RR, FC
1. Physical		RR	RR

Whenever a corrupt or derouted packet is detected, it is dropped. We distinguish between the integrity of the packet's routing data and the integrity of the payload. The routing data's integrity is checked on a hop-to-hop basis, the payload's integrity is checked on an end-to-end basis, since the payload is only relevant upon its delivery.

The resilient router design is responsible for limiting the effects of soft errors in time, ensuring that resilience and predictability are satisfied. Whenever an error affects a component in the router, its resilient design ensures that the component will recover in a bounded period of time.

The reliable transport of data is then responsible for guaranteeing the packet delivery and integrity, since packets may be dropped due to errors. The reliable transport is flexible. It can be implemented to operate transparently in the transport layer or explicitly in the layers above. An example of the latter is a hardware component such as the Direct Memory Access (DMA) controller, which can implement its own protocol. In the sequel, we detail each mechanism.

4.1 Fault Containment

To contain the propagation of an error-affected flit, the router is equipped with ingress filters in its input ports, as shown in Figure 3. The filter is responsible for deciding whether the flit is valid and may be safely propagated or not. If not, the error-affected flit is dropped before altering the router's state. It executes in parallel with the existing route management logic.

The ingress filter must contain flits affected by errors in the upstream router. This boils down to detecting corruption and derouting, similarly to [22]. At the routers, only the routing data (flit header) is checked against corruption.

4.1.1 Containing corrupt flits

To detect corruption, the flits are equipped with Error-Detecting Code (EDC). EDCs, such as parity bit and Hamming code, differ on their detection capability. CRC is chosen due to its error detection capabilities that cover not only uncorrelated bit flips but also bursts of errors, a typical effect of crosstalk (corruption at the links). We employ the 3-bit CRC generator polynomial 0x5 $= (x^3 + x + 1)$ and the 2-bit 0x2 $= (x^2 + 1)$, which are able to detect a single bit flip or a burst of up to 3 and 2 erroneous bits, respectively [27]. This is sufficient given that the CRC is expected to be checked before a second error occurs. Applying other EDCs or other CRC polynomials at design time is possible but not in the scope of this paper. No configuration registers are required.

Figure 4 shows the CRC bits added to the flits, in the first phit of BF/TFs and in the second phit of HF/SFs. We protect only the flit header because it allows corruption to

be detected without requiring the whole flit to be received and processed. Detection can be performed in parallel with route management and can effectively prevent the flit from taking part in the next arbitration cycle without additional delay or stage in the pipeline.

Due to the route management at each router, the CRC code of HF/SFs must be updated accordingly. The new hash is calculated by CRC Gen (see Figure 3) after the route has been updated and only must be ready when the flit is leaving the buffer. When the flit transmission starts, the old CRC hash is overwritten with the new one. BF/TF headers are not modified, thus their CRCs are not updated.

In addition to the cases concerning corruption due to transport, corruption during processing must also be accounted for. Regarding the transmission of an already-forwarded flit, the CRC stored in the buffer must be invalidated after being used, so that, in case the flit is forwarded a second time, it may be detected and contained. Regarding the route management, the CRC must be calculated based on a trusted copy of the updated route.

4.1.2 Containing derouted flits

The detection is not trivial because it must differentiate between a correctly routed flit and a flit forwarded in the wrong direction, despite their correct header routing data. To differentiate them, we check the LOP, the last output port traversed by the flit. Although the principle is the same, let us address HF/SFs first, and then BF/TFs, due to their different formats.

In case of a HF/SF, the flit contains the route encoded as a series of runs (output port and a hop counter). At the input port, the ingress filter will check whether the current input port is the one connected with the output port requested at the last router, the LOP. For instance: the SF's requests the output port "North" (N) and in the next router is processed at the input port "East" (E), which characterizes a derouting, since it should be processed at an input port "South" (S). The input port knows at design time to which output port it is connected, e.g. S connected to N, E to W, and so on. The LOP can be found in the first or in the last position in the route, due to route rotation, and is determined at run-time.

In case of a BF/TF, the checking works in the same way but a field containing the LOP is added to the flit header (see Figure 4). The field is updated at each hop (similarly to the above-mentioned CRC update): the LOP stored at the input buffer overwrites the field in the flit when the flit is being transmitted. The flit's LOP field does not need to be covered by the CRC because, in the event of an error corrupting the LOP field, it will not be valid at the next input port and the flit will be dropped. This way, the CRC of the BF/TF remains unchanged along the route. Notice that the correct operation of the mechanism requires the integrity of the LOP stored at the input buffer, which can be achieved e.g. by storing the value twice, one reserved for the flit header and the other for the actual routing purpose.

Let us now address the two cases regarding the uplinks. When the flit goes from the network interface into the network, no checking is necessary, since no derouting is possible so far. When the flit leaves the network and reaches the network interface, the network interface checks whether the last requested port is equal to its port number.

4.2 Resilient Router Design

To achieve a resilient router it is necessary to eliminate scenarios where soft errors lead to static effects. Although we discuss the architecture, the implementation plays an important role to achieve resilience. Therefore, we assume that the components are implemented in a way that they continue responding in each arbitration cycle independently from the success of the previous arbitration cycle. We now address each component separately.

4.2.1 Pre-processing

The pre-processing contains the route management and the newly added CRC generator and input filter, discussed in the previous section. A transient fault in the pre-processing may cause flit corruption and derouting. The effects are transient since the state is reset at the arrival of each flit.

4.2.2 Input Buffer

The component is responsible for receiving and storing flits, and in parallel, interacting with the arbitration logic and forwarding flits. A transient fault in the input buffer may cause transient flit corruption, loss, and derouting.

Soft errors cause static effects in this component when implementing the buffer as one memory (Data Buffer in Figure 3). This is the case, for instance, when optimizing the design for Field-Programmable Gate Arrays (FPGAs). In such a design, the virtual channel queues are inside the memory and each queue is managed through its read and write pointers. The data required for routing decisions (e.g. output port, flit type) are also kept in a queue at the Control Buffer due to the limited access to the memory. The static corruption then happens when the control and data buffers have two pointers and an error causes them to desynchronize or to continuously access the memory with a wrong offset. To prevent this, the Control Buffer must store the pointer and the Data Buffer derives its pointer from that one when required. Similar care is required when handling the flit as phits. In addition, flits received when a virtual channel queue is full must be dropped while keeping the current queue data and state unaltered.

4.2.3 Crossbar Switch

The crossbar switch connects a given input buffer to an output port. The selection is configured by the switch arbiter. A faulty crossbar causes flit corruption, loss, and derouting. The effects are however transient, since the state is reset at each arbitration.

4.2.4 Switch Arbiter

The arbiter has two stages [25]. The first stage arbitrates in parallel, for each output port, one input buffer that requests access. The second stage, at each input buffer, arbitrates a flit from one of the virtual channels that received a grant from the first stage. Each stage is implemented as a Round Robin scheduler. For wormhole switching, the arbiter is also aware of the virtual channel reservation states managed by the VCAC.

An error affecting the arbiter may cause flit derouting, flit loss, and priority loss. Flit derouting and loss occur when an error corrupts the arbiter when a grant is being accepted

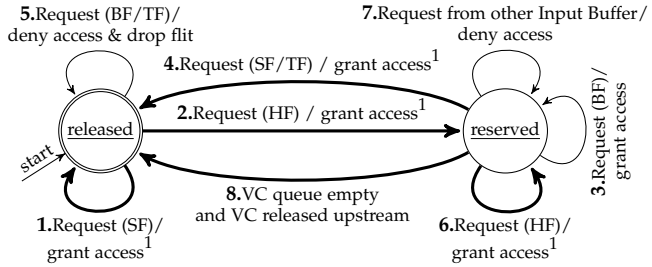


Fig. 5. The Resilient VC Flow Control: Mealy state machine.

or when it is configuring the crossbar. Those effects are guaranteed to be transient because the grant and configuration data is new at each arbitration cycle. Priority loss occurs when an error affects the state of a round robin arbiter.

A faulty Round Robin may cause priority loss. To ensure that this priority loss is transient (i.e. transient blocking instead of static), the arbiter must detect invalid priorities and advance to a valid one. Encoding priorities (Most Recently Served) as one-hot already serves the purpose. Then, when no bits or more than one bit is "hot", the priorities are reset to a initial state, ensuring a resilient arbiter.

The same applies to priority-based arbiters, where VCs have different priorities. In this case, an error affecting the arbiter may also cause flit derouting, flit loss, and priority loss. Those effects are transient since the grants and requests per priority are new at each arbitration cycle. However, depending on the tie-breaking policy implemented by the arbiter, e.g. round-robin arbitrates requests of same priority to the same output port, the considerations above must also be taken into account.

4.2.5 Virtual Channel Access Controller

The VCAC manages the access to virtual channels at each output port. Due to wormhole switching, an input buffer has exclusive access to a VC at an output port, creating a "hole" that starts with the first flit of the packet and closes with the last one (the "worm"). A fault affecting the VCAC then leads to an improper VC release or an improper reservation. Both may provoke static blocking at the router. An improper VC reservation leads to packet blocking ranging from transient to static because it cannot be detected without knowing the state of the upstream routers or until the arrival of a new packet from a certain direction, which is not guaranteed.

To prevent blocking effects, we propose a *resilient virtual channel flow control*. Figure 5 presents the state machine, which is an extension of [6]. The extension is depicted with bold lines. Transitions are in the Mealy format: *condition/output*. Detecting an improper VC reservation requires either a timeout or knowing the state of the upstream router. Since timeouts for managing virtual channels are not really an option[2], we adopt additional wires between routers. The scheme requires one wire per VC that implements wormhole switching to inform a router of the VC reservation state of the upstream router.

1. Transitions 1, 2, 4 and 6 also release in all output ports reservations of the respective VC to the input buffer in question.

2. For instance, finding a suitable timeout value, which depends on the traffic. Values too large result in high latencies; values too small result in packet loss/corruption.

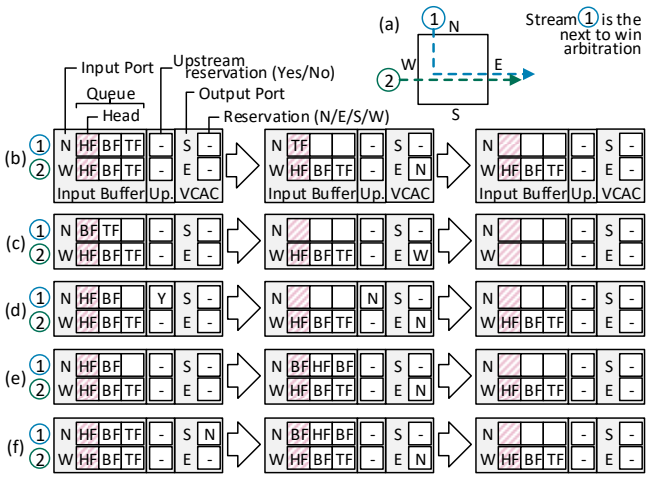


Fig. 6. Examples of resilient VC reservation handling, considering one VC and two streams traversing the router as shown in a. b: regular operation; c: improper release; d,e,f: improper reservation.

The regular VC reserve/release operation in Figure 5 comprises the transitions 1 to 4, as well as transition 7, which refuses access to a VC reserved to another input buffer. An example is given in Figure 6b: first, packets in input N and W request access to the same VC in output E; N wins the reservation first and its flits are being transmitted (trans. 2 and 3) while the packet in W waits (trans. 7); finally, the transfer is completed, the VC is released (trans. 4), and W may continue. In addition to the regular behavior, the state machine specifies the behavior for faulty scenarios, which are divided in two categories: improper release and improper reservation.

The "improper release" represents cases where flits without routing data are at the head of the VC queue at the input buffer and no VC is reserved for it. These are handled by transition 5, which signals the input buffer to drop the flit. Examples are head-less packets and early VC release caused by a transient fault. The former is shown in Figure 6c, where the flits of the head-less packet in N are discarded.

The "improper reservation" represents cases where the VC is not correctly released. Two situations may occur: the queue at the input buffer for the improperly reserved VC is either empty or not empty. Transitions 4 and 6 handle the case where the queue is not empty: a HF/SF tries to reserve a VC and it is already reserved for his own input buffer, the reservation is handed over to the new packet and the router resumes regular operation (trans. 3 or 4). Transition 8 handles the case where the queue is empty: the VC is released if it is empty and the VC upstream is released. If the VC is reserved upstream, no assumptions can be made since the rest of the packet may be still coming. Figures 6d and 6e show examples for transitions 8 and 6, respectively. In 6d, a tail-less packet in N is forwarded and the queue is empty afterwards; when the upstream reservation is released, the respective VC is also released. In 6e, a tail-less packet in N is followed by a second packet, which takes over the reservation (trans. 6) and resumes regular operation. In addition, transitions 1, 2, 4 and 6 must also release the VC in all output ports where they are reserved for the same input buffer. Figure 6f shows an example for such a case, where a VC is improperly reserved in output port S. The reservation

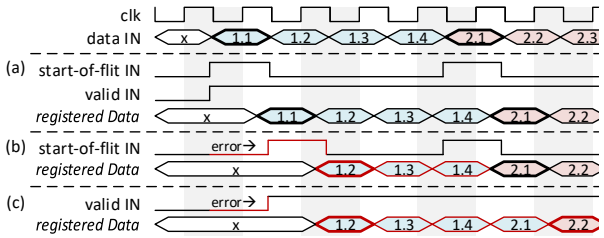


Fig. 7. Comparison of link input signals and the registered data using start-of-flit and valid signals. a: regular operation; b,c: error cases.

is released by the packet in N (trans. 2) before resuming regular operation.

With the proposed scheme, transient faults will result only in transient effects since they will last only until one of the transitions 1, 2, 4, 5, 6 or 8 occur.

4.2.6 Link

The link has control and data signals which can be affected by faults. Faults affecting the data signals cause flit corruption, which are detected at the ingress filter of the next router. Faults in the control signals may cause one-flit loss or blocking lasting one arbitration cycle, depending on whether the valid signal or the flow-control is affected. To prevent multi-flit loss due to synchronization issues when transmitting flits as phits, a start-of-flit signal must be employed instead of a simple valid signal. In the former, the signal is enabled only when transmitting the first phit of a flit, while in the latter the signal is enabled during the transmission of all phits.

The concept is illustrated in Figure 7, where a flit f is composed of 4 phits ($f.1$ to $f.4$) transmitted subsequently on the Data bus. The first phit of a flit is highlighted with thick lines. In regular operation, Figure 7a, start-of-flit and valid signals have the same effect: the flit is received as intended by the sender, the *registered data*. In case of an error causing the start-of-flit to be set (or propagated) high one cycle later, only the affected flit will be lost when transmitting flits back-to-back. This is seen in 7b where the first received flit starts with the second phit but the subsequent flit is received correctly. However, in the case of the valid signal, it is not possible to identify the end of a flit and start of the next one. This is seen in 7c, where first received flit starts with the second phit and includes the first phit of the second flit, since the receiver expects four phits after the valid signal is first set high. The same occurs to the subsequent flits, resulting in the loss of all flits transmitted back-to-back on that link after the error. Corrupt flits are discarded (cf. Section 4.1 and Section 4.4).

Faults affecting the VC reservation signals introduced above may cause the loss of a packet when it enables the transition 8 in Figure 5.

4.3 Between Lower and Upper Layers

Before presenting the reliable transport, let us discuss what has been achieved with the hardening so far. As summarized in Table 2, the proposed fault containment ensures: *integrity* of the data utilized in the lower layers of the NoC, i.e. the routing data; and part of the *real-time* requirement as it prevents the error from propagating to other traffic

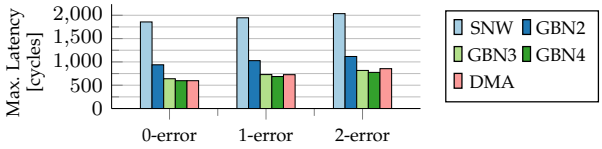


Fig. 8. Maximum latency of an 8kB DMA on different ARQ protocols [7].

streams and to other routers (the "background traffic" in [6]). The resilient router design ensures the *resilience* and the remaining part of the *real-time* requirement by ruling out static effects and providing for a recovery limited in time.

Let us summarize the behavior of the network *before* applying the reliable transport. On the end-to-end perspective at the network interfaces, verifiable with [6], a packet sent through the NoC and affected by an error is either:

1) delivered correctly,
2) delivered correctly with a small delay l^{rec},
3) delivered with corrupt payload,
4) or dropped/lost.

The reliable transport is responsible for guaranteeing the packet delivery and guaranteeing the integrity of the transported data, i.e. the payload.

4.4 Reliable Transport

Aiming at flexibility, our approach provides guaranteed delivery and payload integrity on an end-to-end basis. Therefore, the reliable transport is implemented in the transport layer or above. It relies on EDCs or ECCs for error detection and possibly correction and relies on protocols, such as ARQ and multipath routing, for guaranteeing packet delivery. The approach enables the system controller or application to configure at runtime in the network interface the appropriate configuration (protocol+EDC/ECC) for each transaction according to its characteristics, reducing unnecessary overheads, both in terms of traffic and power consumption. For instance, a DMA transfer may use its own optimized protocol, while a command to an actuator could use Stop-and-Wait ARQ. When a retransmission cannot be afforded due to a deadline miss, multipath routing can be employed. Besides, periodic sensor readings may require only integrity guarantee with EDC/ECC and no delivery guarantee. Moreover, the diverse traffic may co-exist in the NoC.

Formal guarantees for ARQ-based protocols operating in the transport layer of wormhole-switched NoCs have been introduced in [7], an extension of [13]. The work provides formal analysis for three protocols: Stop-and-Wait, Go-Back-N, and DMA ARQ, an optimized protocol for DMA transfers. Formal guarantees for other protocols, such as multipath routing, can be similarly derived. In [7], the authors show that the overhead introduced by the protocols' handshaking is acceptable and they perform very well with typical real-time on-chip traffic, provided that the appropriate configuration is employed. As an example, the latency of an 8kB DMA transfer when using different protocols and configurations is plotted in Figure 8 (extracted from [7], where the setup is described). In the figure, in the error free case (0-error), the handshaking overhead of ARQ is almost negligible for Go-Back-N with N=4 (GBN4) and the DMA ARQ (DMA).

To provide minimum performance guarantees, formal analyses assume a given number of errors that the system may experience in a given time interval, the so called k-error scenario [7], [13]. In the NoC, it translates to errors affecting a given packet or transmission. k depends on the Bit Error Rate (BER), to which the NoC is subject, and can be calculated using methods in [28]. k must be selected so that the probability that the packet or transmission experiences more than k errors is negligible according to compliance levels of safety standards [2], [3]. The k-error scenario is then used to calculate the worst-case latencies and response times of the system under errors. Usually, at most one error will be considered since the probability of two errors within a hundred clock cycles is negligible. Naturally, the final latencies of the NoC depends on the specific protocol. In Figure 8, latencies in scenarios with 0, 1 and 2 errors are shown. Error occurrences imply slightly longer latencies due to retransmissions and the timeout period. Nonetheless, that does not prevent their use in a real-time system. Alternatively, the error case can be modeled as an overload scenario in typical worst-case response time analysis [29].

Note about the delay l^{rec}: when employing retransmission protocols, it is not necessary to know the exact value of l^{rec}. The formal guarantee under errors only requires l^{rec} to be smaller than the time to retransmit the packet [7].

5 Experimental Evaluation

We evaluate the resulting NoC's reliability, performance under errors and implementation overhead when compared to the baseline NoC. The performance under errors is evaluated with synthetic random traffic as well as a real-world use case. We employ a NoC in a 2D-mesh topology with different sizes, 5 VCs, and one up-link per router. From the reliable transport layer, we employ only EDCs to ensure the payload integrity.

The objective of the experiments is to evaluate the predictability and reliability of the NoC under soft errors. Evaluating the impact of bit-flips in the NoC or the performance of ARQ-based protocols is not our goal. Transport protocols have well know properties and can be applied on top of the resulting NoC. On that topic, the interested reader can refer to [7], [13]. On the impact of soft errors in NoCs, the interested reader can refer to the reports of [6], [11], [23].

5.1 Reliability

Let us evaluate the reliability metric $\mathcal{R}(t)$, which is the probability that the NoC does not fail during a time interval $[0, t]$ [28]. In a high dependability real-time system, the failure is defined as the violation of integrity, resilience or real-time latency guarantees (which include static effects leading to blocking) due to errors. Packet loss is not considered as a failure, since it is handled in the transport layer. In practice, expected BERs[3] are in the order of 10^{-9} bit-flips per hour [31]. Moreover, the design must however consider higher rates (up to 10^{-6}/hour) as a safety margin [33]. Here, we consider BERs from 10^{-9} up to 10^{-6}/hour. Additionally, we consider a permanent fault rate of 10^{-8}/h per router[4]

3. We derive [30] the BERs for seq. and comb. logic from [31] for 65nm CMOS SRAM. Masking effects [32] are not taken into account.

4. The occurrence of a permanent fault leads directly to failure. The fault rate per router is derived from processor failure rates in [34].

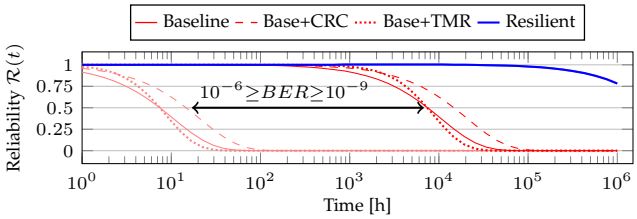


Fig. 9. Reliability comparison of the proposed NoC and non-resilient ones. A 5x5 NoC size is considered.

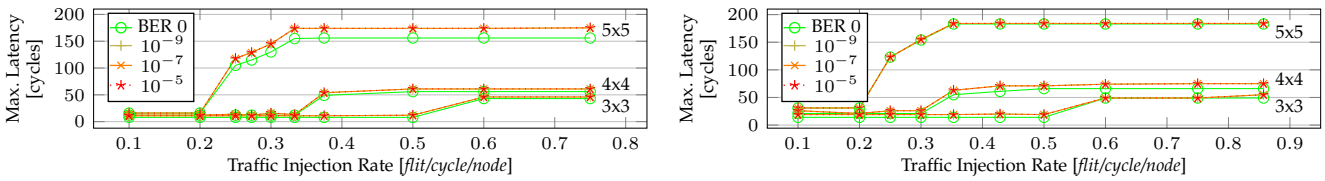
Figure 9 plots the analytical $\mathcal{R}(t)$ for the non-resilient baseline and the proposed NoCs (resilient), considering a 5x5 2D-mesh topology. Two variants of the baseline NoC with CRC checking (Base+CRC) and with Triple Modular Redundancy (Base+TMR) are also plotted. The TMR is non-reparable with an ideal voter [28]. We can observe that, independent from the BER, regular non-resilient NoCs are limited in time due to errors from which they cannot recover. Employing ingress filters (Base+CRC) improves the reliability but it continues limited. Moreover, contrary to common sense, triplicating the NoC does not make it more reliable with respect to soft errors. In time, the more-than-double area overhead of the TMR has more impact than the initial advantage of withstanding one error. When considering TMR with periodic reboot, e.g. weekly, the reliability would reach 0.962 (5x5) and 0.907 (8x8) before the reboot (BER 10^{-9}). The proposed hardening enables the NoC to drastically increase the reliability in time by appropriately containing soft errors and recovering from them.

Let us evaluate the Failure In Time (FIT). The metric measures the number of failures in a billion hours. Table 3 reports the FIT rates for different NoC sizes and BERs. Even in small topologies, non-resilient NoCs present very high failure rates. Besides, TMR leads to more failures than no redundancy. To put the values in perspective, high dependability systems, e.g. in the automotive domain, must present less than 10 random hardware failures in time when implementing critical functionality with SIL 4, the highest safety integrity level [4]. In a system providing a less critical functionality with SIL 1, the lowest integrity level, up to 10^4 failures in time are acceptable [4]. Notice that a final FIT rate for the MPSoC must consider other components, resulting in even larger FITs. As a component of the final system, the MPSoC requires a dependable and resilient NoC.

Despite the NoC's high reliability, errors still have an impact on the traffic latency while the routers recover from them. This impact is evaluated next.

TABLE 3
Comparison of FIT rates

	BER/h	3x3 NoC	5x5 NoC	8x8 NoC
Baseline	10^{-9}	$3.38 \cdot 10^4$	$9.40 \cdot 10^4$	$2.41 \cdot 10^5$
	10^{-6}	$3.38 \cdot 10^7$	$9.38 \cdot 10^7$	$2.4 \cdot 10^8$
Base+CRC	10^{-9}	$1.72 \cdot 10^4$	$4.78 \cdot 10^4$	$1.22 \cdot 10^5$
	10^{-6}	$1.71 \cdot 10^7$	$4.75 \cdot 10^7$	$1.22 \cdot 10^8$
Base+TMR	10^{-9}	$4.06 \cdot 10^4$	$1.13 \cdot 10^5$	$2.89 \cdot 10^5$
	10^{-6}	$4.05 \cdot 10^7$	$1.13 \cdot 10^8$	$2.88 \cdot 10^8$
Resilient	10^{-9}	$9.0 \cdot 10^1$	$2.5 \cdot 10^2$	$6.4 \cdot 10^2$
	10^{-6}	$9.0 \cdot 10^1$	$2.5 \cdot 10^2$	$6.4 \cdot 10^2$



(a) 3-flit packets

(b) 6-flit packets

Fig. 10. Observed worst-case NoC performance under random uniform traffic as the load increases, varying NoC sizes, packet sizes and error rates.

5.2 Performance under errors: random traffic

Let us now evaluate how the proposed NoC behaves under errors. The performance is evaluated through error injection experiments carried out in the OMNeT++ simulator [35]. For that, uniform random traffic consisting of 3-flit packets was injected in the NoC, with the injection rate varying with the experiment. The evaluation considers a clock period of 2ns and a transmission time of 1 clock cycle per flit. The errors from Table 1 were injected randomly into the NoC[5]. The occurrence probability of an error depends on the affected data structure and the BER. The BERs employed range from 10^{-9} to 10^{-5}/h.

In the first experiment, faults are injected randomly into the whole NoC. Figure 10 shows the observed worst-case performance of the proposed NoC, as the load increases, for different NoC sizes, packets sizes and BERs. Performance degradation is minimal even when experiencing high error rates. By design, errors also cause packets to be dropped when corrupted or derouted. On average, $1.11 \cdot 10^{-14}$% of the packets were dropped under BER 10^{-5}, $1.11 \cdot 10^{-16}$% under 10^{-7}, and $1.11 \cdot 10^{-18}$% under 10^{-9}.

The impact of errors on latency in the hardened NoC depends on the number of errors that affect a packet in the network and on the current load of the router where the error occurs. The load depends on the mapping, the routing algorithm and indirectly on the NoC size – a larger NoC may present higher loads in central routers, up to the maximum load a router is able to handle. This can be seen in Figure 10a, where maximum latencies[6] both in the error free and in the error cases increase with the load until congestion occurs in the network, e.g. 0.33 in 5x5 NoC. Moreover, the results show that the performance degradation is predictable even under very high load and BER. Increasing the packet size does not have major impact on the latency under errors. This is confirmed by Figure 10b, which shows the NoC performance when doubling the packet size (6-flit packets).

In a second experiment, we evaluate the error propagation between affected and unaffected VCs. Therefore, errors are injected only in the central router and its links. We then differentiate affected traffic stream, whose packets traverse the faulty router, and unaffected streams, on a VC whose traffic do not traverse that router. It is observed that error effects do not propagate between different VCs as the latency, integrity and delivery of packets in unaffected VCs are not affected. Considering that different traffic classes and criticalities are allocated to different VCs, this means that sufficient independence is achieved in the presence of errors. The same behavior is observed across different network sizes.

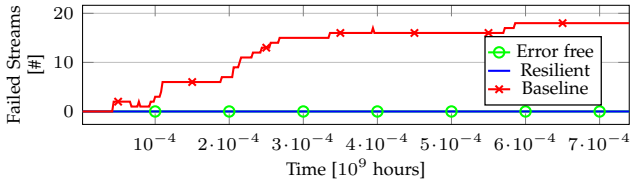
In a third experiment, we evaluate the performance in time of the proposed NoC and compare it with a baseline one as errors are injected randomly in the whole NoC. The results are reported in Figure 11, where each point plotted represents a time interval of two thousand hours. Figure 11a shows the number of failed streams in time, from a total of 25 traffic streams in the NoC. The same definition of failure from Section 5.1 is used. The resilient NoC is able to recover from errors, as it ensures that soft errors have only transient effects. In contrast, the baseline NoC is not able to do so and traffic streams are blocked when certain errors occur. Notice that not all errors lead to a stream failure (cf. Section 3.2). Packet delivery is reported in Figure 11b. The metric consists of the overall number of packet delivered among all traffic streams in the NoC relative to the error free case (100%). With time, the baseline NoC delivers less packets as a result of blocking and static effects. In the resilient NoC, on the other hand, packets are delivered continuously albeit with expected loss ($1.39 \cdot 10^{-15}$%). The maximum latency in time observed in the resilient and baseline NoCs are plotted in Figures 11c and 11d, respectively. The latencies are given for each traffic stream, 25 in total (some lines overlap). The predictability of the proposed NoC is seen as the maximum latency varies seldom and within a limited range. In contrast, the unpredictable behavior of a non-resilient NoC presents very high latencies or blocking (latencies equal 0). Moreover, Figures 11c and 11d shows the observed experimental trend that was previously only illustrated in Figure 2.

Let us now compare the performance of the resilient NoC with TMR. Differently than in our approach, the latency and packet delivery do not vary with or without errors in a NoC with TMR configuration. However, that holds as long as the TMR itself does not fail, i.e. 2-out-of-3 instances survive. After failure, depending on the voter, the same behavior of the baseline NoC is observed.

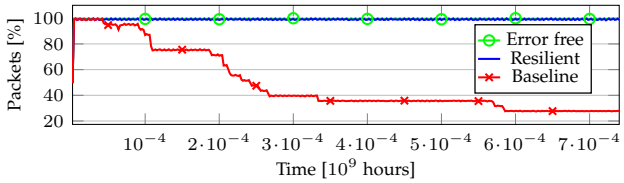
In addition to the presented experiments, the proposed NoC has been evaluated in many other experiments (thousands), where we vary the packet sizes, traffic patterns, network sizes, and error rates. The experiments employed benchmarks as well as randomly generated mappings and loads, the latter as an effort to stimulate worst-case scenarios. In this paper, only some results are reported (>120). Compared to these, similar or better performance (i.e. no worse case) was observed across all experiments (for different number of flits and traffic patterns).

5. To speed-up simulation, errors were injected in active areas without impairing the evaluation.

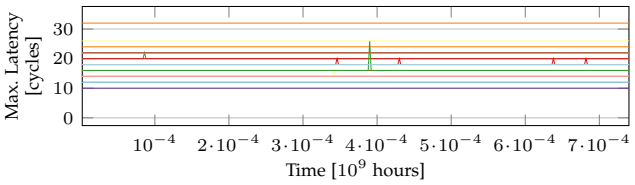
6. Latency does not comprise retransmission (not under evaluation) or the time spent in the network interface (in case of congestion), only the time spent inside the network.



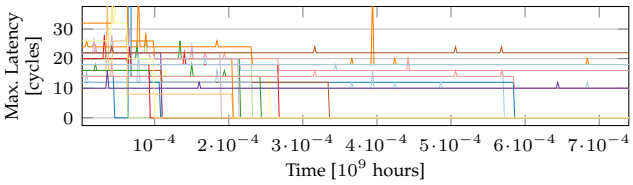
(a) Failure of traffic streams in time



(b) Overall packet delivery in time



(c) Maximum latency in time per traffic stream: resilient NoC



(d) Maximum latency in time per traffic stream: baseline NoC

Fig. 11. Performance in time of the proposed NoC (resilient) and a non-resilient one (baseline). 5x5 NoCs, 3-flit packets, traffic injection rate 0.2 (flit/cycle/node) and BER=10^{-6}.

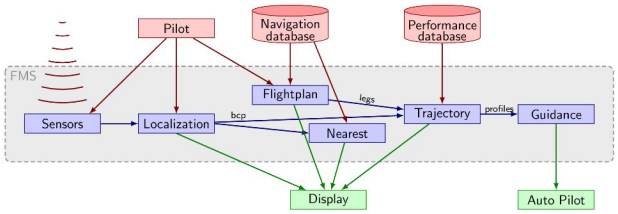


Fig. 12. Overview of the FMS application [36].

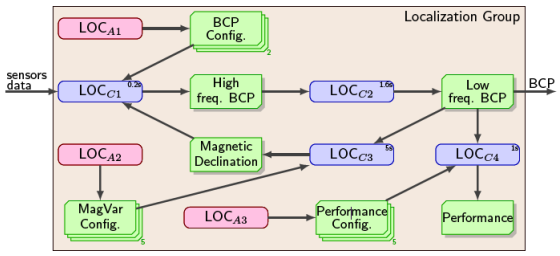


Fig. 13. FMS: Localization task group [36].

5.3 Performance under errors: FMS use case

Let us now evaluate the NoC with the FMS as a realistic use case. The Flight Management System is a high dependability embedded application in modern avionics that automates several in-flight tasks. It is widely found in civilian as well as in military aircraft. A functional overview of Thales' FMS [36] is shown in Figure 12 (arrows indicate the data flow). The application consists of 6 task groups: *Sensors*, *Localization*, *Flightplan*, *Nearest*, *Trajectory* and *Guidance*. The tasks in the groups are executed periodically, when in automatic mode. Some of them can also be triggered aperiodically through a manual intervention by the pilot. In our evaluation, we focus on the *Localization* task group.

The *Localization* task group is responsible for periodically computing the Best Computed Position (BCP), the most probable position of the aircraft, using data from sensors, such as Global Positioning System and Pure Inertia Reference System, received from the Sensors task group. The task group is detailed in Figure 13 (arrows indicate the data flow). The BCP computation also must account for settings that can be modified by the pilot, captured in Figure 13 by the aperiodic tasks LOC_{A1}, LOC_{A2} and LOC_{A3}. The execution follows the Acquisition-Execution-Restitution task model, where at the end of each execution, the output of a task is written to the dependent tasks' buffers [36]. The FMS has, among others, a requirement at the application level stating that a task can execute at most 3 times without receiving new valid input data. In that case, the last valid input data is used. After that, the task enters an error mode, i.e. fails. Corrupt data is discarded and not considered as valid.

When mapped to an MPSoC, the tasks' outputs are transferred with traffic streams over the NoC. Table 4 list the traffic streams associated with the periodic tasks of the *Localization* task group. For instance, the output of LOC_{C1} is transferred to LOC_{C2}'s buffer through stream $C1 \rightarrow C2$ every 200ms on a 11-flit-long packet. Since the application tolerates data loss, the traffic streams only employ EDCs to guarantee integrity in the reliable transport, i.e. do not employ a reliable transport protocol to guarantee delivery. For other cases, such as application initialization, e.g. through DMA, which might require guaranteed data delivery, the interested reader can refer to [7].

Let us now assess the failure probability of the *Localization* task group due to soft errors in the NoC. The evaluation uses a BER of $10{-}6$/h and different NoC sizes, safely assuming that each stream traverses the longest route in the NoC. The wost-case packet latencies are obtained from the interference they experience in the NoC and the application activation patterns [24]. The failure probability in the resilient NoC equals to the probability that three subsequent packets are lost or corrupt while inside the NoC. Due to the NoC's properties, such as fault containment, and

TABLE 4
Localization computing tasks performance under BER 10^{-6}/h

Traffic Stream	Period [s]	Size (flits)	Failure probability		
			3x3 NoC	5x5 NoC	8x8 NoC
$C1 \rightarrow C2$	0.2	11	$1.31 \cdot 10^{-39}$	$2.57 \cdot 10^{-39}$	$5.64 \cdot 10^{-39}$
$C2 \rightarrow C3$	1.6	11	$1.31 \cdot 10^{-39}$	$2.57 \cdot 10^{-39}$	$5.64 \cdot 10^{-39}$
$C2 \rightarrow C4$	1.6	11	$1.31 \cdot 10^{-39}$	$2.57 \cdot 10^{-39}$	$5.64 \cdot 10^{-39}$
$C3 \rightarrow C1$	5	3	$3.37 \cdot 10^{-42}$	$1.13 \cdot 10^{-41}$	$3.81 \cdot 10^{-41}$
Task group overall:			$3.94 \cdot 10^{-39}$	$7.71 \cdot 10^{-39}$	$1.69 \cdot 10^{-38}$

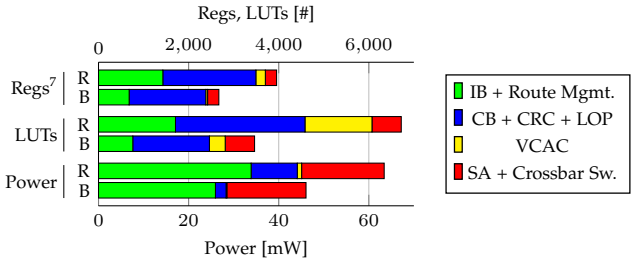


Fig. 14. Synthesis results for a 5-port router of the baseline NoC (B) and the resilient NoC (R).

as shown in previous experiments, it suffices to consider errors directly affecting the packets under evaluation, which include the packet contents and registers critical to packet transmissions in the network. The results per stream and per task group are shown in Table 4. The very low failure probabilities per stream and also for the whole task group show that this type of traffic can be safely transported in the network without the use of transport protocols even under high BERs. It results in less traffic in the network, since there is no overhead from transport protocols, and consequently in lower power consumption. This type of traffic is also typically seen in transmissions of periodic sensor readings.

5.4 Implementation overhead

Let us now evaluate the implementation cost to achieve the results above. The NoC was implemented in VHDL and synthesized in Xilinx ISE targeting a Virtex-6 FPGA (xc6vlx760) and a frequency of 100MHz. We report the data for a 5-port router of the proposed and the baseline NoCs. The router size increases 5.39% when accounting for all the data in the router (optimized for FPGAs, each data buffer instantiates a Block RAM). Figure 14 details the resource usage and total power for the router and its internal components. In the figure, the input buffer is divided into two categories: the control buffer and ingress filter (CB+CRC+LOP) and the remaining components of the input buffer (IB). The 5 BRAMs are not included in the figure.

The implementation overhead of the proposed NoC when compared to the baseline is caused mainly by the ingress filter and by the resilient VC flow control. Indeed, the ingress filter (CB+CRC+LOP) is responsible for most of the additional register bits (43.73%) and 36.06% logic (LUTs). However, such filter is a standard component in many resilient approaches, such as [20], [22], and can be considered as a necessary baseline resiliency cost. The overhead specific to our approach is introduced by the resilient VC flow control (VCAC), which requires additional wires in the links and control logic, corresponding for 10.40% additional registers and 34.92% logic. Other minor overheads are caused e.g. by the packet-dropping logic in the IB. In addition, we evaluated the energy consumption with Xilinx Power Analyzer [37]. Under full load (random traffic with random payload), the resilient router consumes 37.68% more energy than baseline (cf. Figure 14). When idle (no traffic), the overhead is 0.28% (not shown). Under regular operation, the router load is expected to be closer to the latter than to the former.

7. Not including the data buffers, synthesized as Block RAMs.

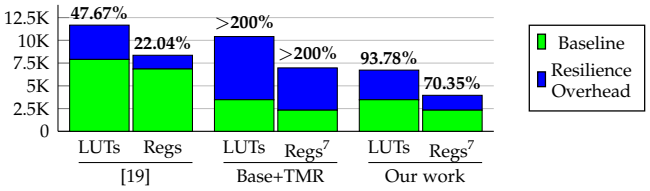


Fig. 15. Comparison of resilience overhead per router (Virtex-6 FPGA).

Finally, in Figure 15, we compare the overhead of our approach with TMR and related work [22] (although the two approaches are not equivalent – cf. Section 2). Let us first make two considerations about Figure 15. First, our baseline router is very lean, implemented with the minimal functionality required. Second, the synthesis tool optimally maps the data buffers (28K bits) to Block RAMs of the FPGA, leading to an apparent large overhead w.r.t. register usage. Thus, our router stores over 17 times more data than [22]'s. Nonetheless, our absolute overhead when synthesized to the same Virtex-6 FPGA family is similar to [22]'s. Considering all data in the router including data buffers, our relative overhead is only 5.39% as opposed to [22]'s 22.04%. In comparison to TMR, not only is our approach much more efficient w.r.t. resource usage (area) and power but also more effective (cf. Section 5.1). We are able to achieve predictability and much higher reliability with only a fraction (5.39%) of TMR's total relative overhead of >200%. Moreover, we require at most 37.68% (in a traffic peak) additional power while TMR has a constant twofold power overhead. Besides, TMR implies a substantial increase in interconnecting wires, leading to design routing complications, potential congestion and lower frequencies.

6 Conclusion

In this paper, we have presented a resilient wormhole-switched NoC hardened against soft errors. As opposed to the state of the art, the NoC aims at the integrity, resilience and real-time requirements of high dependability systems. At the same time, it aims at preventing the occurrence of silent data corruption, a challenge in such systems. Therefor, the approach takes into account all possible durations and impacts of soft errors. The errors are detected and handled by three mechanisms distributed in several layers of the network stack. For the sake of predictability and integrity, error-affected packets are dropped and guaranteed packet delivery is selectively provided on an end-to-end basis, with formal timing guarantees. Our experimental evaluation includes an industrial avionics use case and shows that, even under very high error rates, the NoC presents a predictable behavior with an acceptable hardware overhead.

References

[1] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "IDAMC: A Many-Core Platform with Run-Time Monitoring for Mixed-Criticality," in *Proc. of HASE'12*, 2012.
[2] "ISO 26262: Road vehicles – functional safety," International Standards Organization, 2011.
[3] "DO-254: Design assurance guidance for airborne electronic hardware," RTCA Incorporated, 2000.
[4] "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, ed.2.0," International Electrotechnical Commission, 2010.

[5] L. Benini and G. De Micheli, "Powering networks on chips: Energy-efficient and reliable interconnect design for SoCs," in *ISSS*, 2001.

[6] E. A. Rambo, A. Tschiene, J. Diemer, L. Ahrendts, and R. Ernst, "FMEA-Based Analysis of a Network-on-Chip for Mixed-Critical Systems," in *Proc. of NOCS'14*, 2014.

[7] E. A. Rambo, S. Saidi, and R. Ernst, "Providing formal latency guarantees for ARQ-based protocols in networks-on-chip," in *Proc. of DATE'16*, 2015.

[8] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[9] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10–16, Nov 2005.

[10] B. Halak and A. Yakovlev, "Statistical analysis of crosstalk-induced errors for on-chip interconnects," *Computers Digital Techniques, IET*, vol. 5, no. 2, pp. 104–112, March 2011.

[11] E. A. Rambo, A. Tschiene, J. Diemer, L. Ahrendts, and R. Ernst, "Failure analysis of a Network-on-Chip for real-time mixed-critical systems," in *Proc. of DATE'14*, 2014.

[12] A. Tanenbaum and D. Wetherall, *Computer Networks*. Pearson Prentice Hall, 2011.

[13] P. Axer, D. Thiele, and R. Ernst, "Formal timing analysis of automatic repeat request for switched real-time networks," in *Proc. of SIES'14*, Pisa, Italy, June 2014.

[14] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst, "Designing networks-on-chip for high assurance real-time systems," in *Proc. of PRDC'17*, 2017.

[15] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in networks on chip," *ACM Comput. Surv*, vol. 44, 2012.

[16] Q. Yu, B. Zhang, Y. Li, and P. Ampadu, "Error control integration scheme for reliable NoC," in *ISCAS*, 2010.

[17] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 6, 2005.

[18] J. Kim, D. Park, C. Nicopoulos, N. Vijaykrishnan, and C. Das, "Design and analysis of an NoC architecture from performance, reliability and energy perspective," in *ANCS*, 2005.

[19] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. Das, "Exploring fault-tolerant Network-on-Chip architectures," in *Proc. of DSN'06*, 2006.

[20] A. Kohler, G. Schley, and M. Radetzki, "Fault tolerant network on chip switching with graceful performance degradation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 6, 2010.

[21] C. Feng, Z. Lu, A. Jantsch, M. Zhang, and Z. Xing, "Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 6, 2013.

[22] C. Killian, C. Tanougast, F. Monteiro, and A. Dandache, "Smart reliable Network-on-Chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, 2014.

[23] E. A. Rambo, L. Ahrendts, and J. Diemer, "FMEA of the IDAMC NoC," Institute of Computer and Network Engineering – TU Braunschweig, Tech. Rep., 2013.

[24] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *Proc. of DATE'15*, 2015.

[25] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking (TON)*, vol. 7, no. 2, pp. 188–201, 1999.

[26] R. Santos, S. Venkataraman, and A. Kumar, "Scrubbing mechanism for heterogeneous applications in reconfigurable devices," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 2, Feb. 2017.

[27] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *Proc. of DSN'04*, 2004.

[28] A. Høyland and M. Rausand, *System reliability theory: models and statistical methods*. John Wiley & Sons, 2009, vol. 420.

[29] Z. A. H. Hammadeh, S. Quinton, and R. Ernst, "Extending typical worst-case analysis using response-time dependencies to bound deadline misses," in *Proc. of EMSOFT'14*, October 2014.

[30] T. Heijmen, *Soft Errors from Space to Ground: Historical Overview, Empirical Evidence, and Future Trends*. Springer US, 2011, pp. 1–25.

[31] J. Autran, P. Roche, S. Sauze, G. Gasiot, D. Munteanu, P. Loaiza, M. Zampaolo, and J. Borel, "Altitude and underground real-time SER characterization of CMOS 65nm SRAM," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, 2009.

[32] M. Ebrahimi, A. Evans, M. B. Tahoori, E. Costenaro, D. Alexandrescu, V. Chandra, and R. Seyyedi, "Comprehensive analysis of sequential and combinational soft errors in an embedded processor," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1586–1599, 2015.

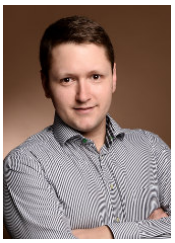
[33] J. Leray, "Effects of atmospheric neutrons on devices, at sea level and in avionics embedded systems," *Microelectronics Reliability*, vol. 47, no. 9-11, 2007, 18th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[34] F. Oboril and M. B. Tahoori, "Exploiting instruction set encoding for aging-aware microprocessor design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 1, pp. 5:1–5:26, Dec. 2015.

[35] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *1st SIMUTools 2008*, 2008.

[36] G. Durrieu, M. Faugere, S. Girbal, D. G. Pérez, C. Pagetti, and W. Puffitsch, "Predictable flight management system implementation on a multicore processor," in *Proc. of ERTS'14*, 2014.

[37] *Xilinx Power Tools Tutorial*, Xilinx Inc., San Jose, CA, 2013.



Eberle A. Rambo received a B.S. and M.S. degrees in Computer Science from the Federal University of Santa Catarina, Florianpolis, Brazil, in 2009 and 2011, respectively. Since 2013, he is pursuing his PhD at Braunschweig University of Technology, Braunschweig, Germany. His research interests include many-core architecture and design for real-time systems with focus on reliability and on Networks-on-Chip.

Christoph Seitz received a B.S. and M.S. degrees in Computer and Communication Systems Engineering from Braunschweig University of Technology, Braunschweig, Germany, in 2013 and 2016, respectively.

Selma Saidi received a Ph.D in computer science from the University of Grenoble in France in 2012. In 2013 She joined the group of Prof. Rolf Ernst in the Technical University of Braunschweig as a Post Doctoral researcher to work on embedded multi-core architectures dedicated to safety critical real-time systems. Since January 2016, She has a permanent research position in the Institute of Embedded System in Hamburg University of Technology. Her research interests currently involve timing analysis for multicore platforms, multi-core ECUs for future cars, Internet of things and new connectivity solutions.

Rolf Ernst Rolf Ernst received a Diploma degree in Computer Science and the Dr.-Ing. degree in Electrical Engineering from the University of Erlangen-Nuremberg, Erlangen, Germany, in 1981 and 1987, respectively. From 1988 to 1989, he was with Bell Laboratories, Allentown, PA. Since 1990, he has been a professor of Electrical Engineering at Braunschweig University of Technology, Braunschweig, Germany. His research activities include embedded system design and design automation.